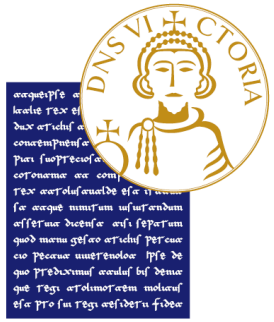
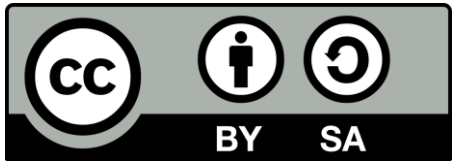


Wedroid

a wekan client for android

Furno Umberto Mignone Raffaele Mincolelli Noemi

Università degli Studi del Sannio
Corso di laurea magistrale in ingegneria informatica
Esami di ingegneria del software



Wekan

Wekan è una *kanban board open-source* attraverso la quale è possibile gestire svariati compiti, come l'organizzazione di To-Do list o la gestione del *workflow* di un team di lavoro.

Wekan oltre ad un *frontend web* mette a disposizione anche un'API rest.

Wedroid

Il nostro gruppo ha deciso di sfruttare l'API di *Wekan* per realizzare una prima implementazione di un client per Android.

Durante lo sviluppo di questo progetto abbiamo fatto ricorso ad una serie di tecniche e strumenti appresi durante il corso.

Tools

Git

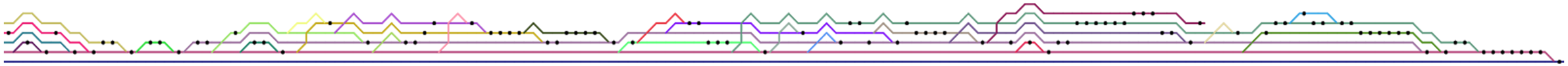
Per lavorare in contemporanea sul progetto e gestire le versione dei vari artefatti abbiamo utilizzato git.

Inoltre abbiamo utilizzato un’istanza personale di ***gitea*** per l’hosting del nostro codice e per la gestione delle code review.

Gitflow

- master: utilizzato per traccia della storia delle *release*
- develop: utilizzato per l'integrazione delle varie *features*
- feature branches: utilizzati per lo sviluppo delle nuove funzionalità
- hotfix branches: utilizzati per la correzioni dei bug
- release branches: utilizzati per la *preparazione* delle *release*

Commit graph



Promozione e rilasci

Master

- Richiesta *green build* del commit
- Richiesta *green build* della *PR*
- Richieste due review positive

Develop

- Richiesta *green build* del commit
- Richiesta *green build* della *PR*
- Richieste una review positiva

Build scripts

L’automatizzazione delle build è avvenuta mediante ***Gradle***.

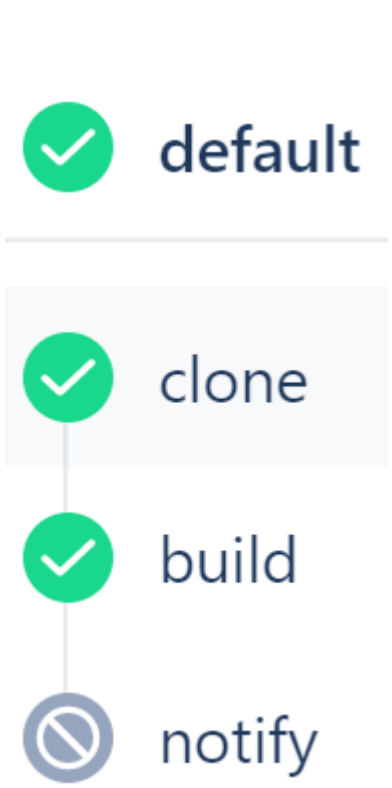
Il progetto è stato diviso in due moduli:

- wrapper
- app

Continuous Integration

Il processo *CI* è stato gestito attraverso un'istanza personale di **Drone CI**.

Pipelines


default
clone
build
notify


release
clone
build
gitea release
notify

Static analysis

Checkstyle

Configurazione personalizzata

realizzata sulla base dello stile di

Google.

Detekt

Configurazione di default di *ktlint*.

La build fallisce se nel codice sono

Presenti almeno dieci *warning*

Static analysis

Standard Checkstyle

```
String board1 =
        "{"
            + "\"_id\":\"id1\","
            + "\"title\":\"my title1\","
            + "}";
```

Moduli modificati:

- OperatorWrap
  - *option = NL* → option = eol
- Indentation
  - basicOffset from 2 to 4
  - caseIndent from 2 to 4

Customize Checkstyle

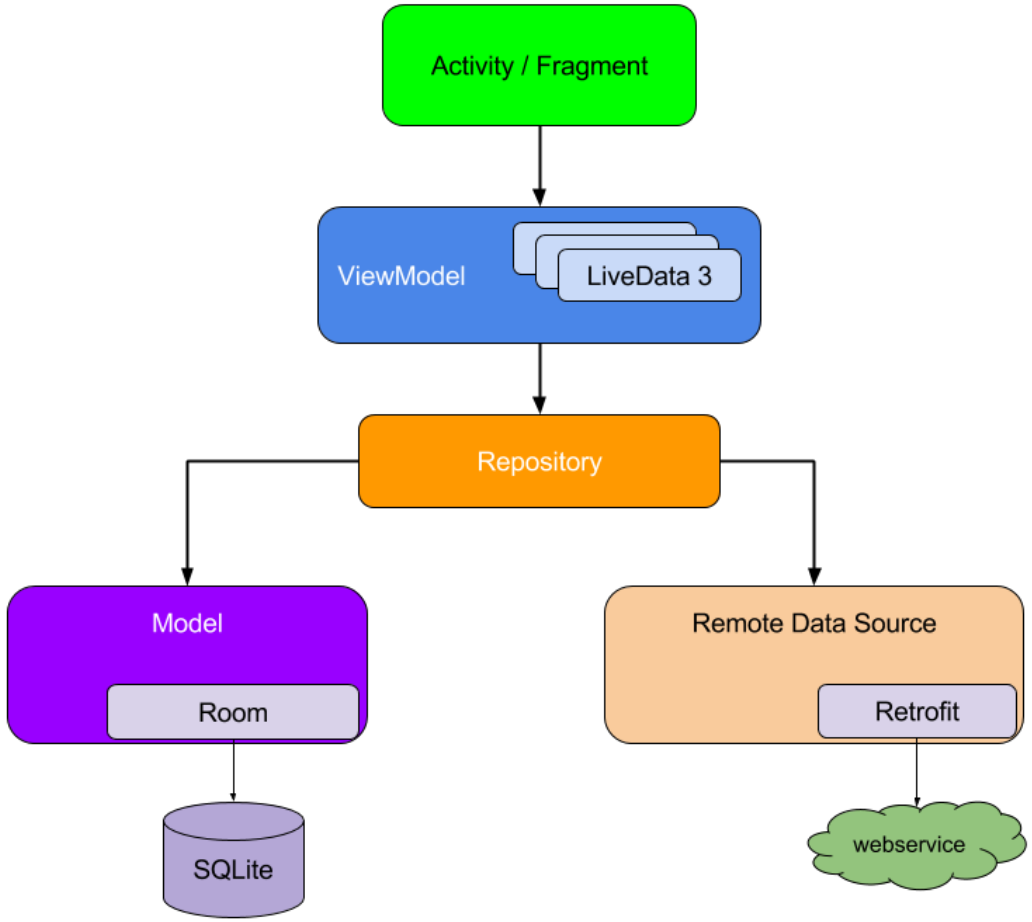
```
String board1 =
        "{" +
            "\"_id\":\"id1\"," +
            "\"title\":\"my title1\"," +
            "}";
```

Moduli eliminati:

- NeedBraces
- JavadocMethod
- MissingJavadocMethod

Wedroid

Architettura



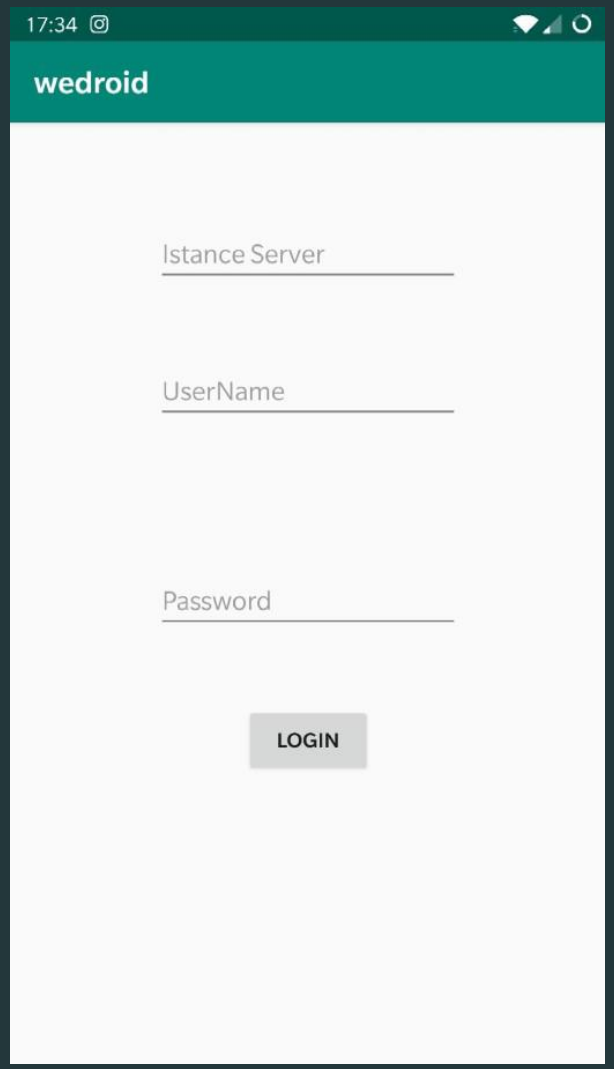
Il *design* dell'applicazione si basa sull'architettura MVVM.

Librerie utilizzate

- Retrofit: Interazione con l'API *rest*
- Room: Layer per l'interazione con il database *SQLite*
- LiveData: Observable data *holder*

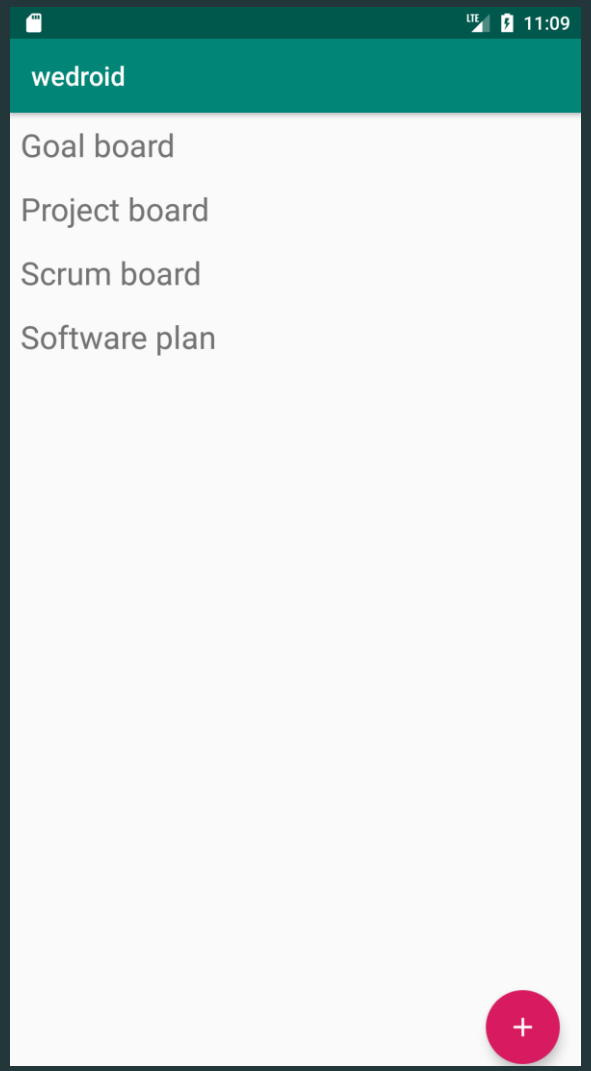
Funzionalità - Login

Possibilità di utilizzare un’istanza personale



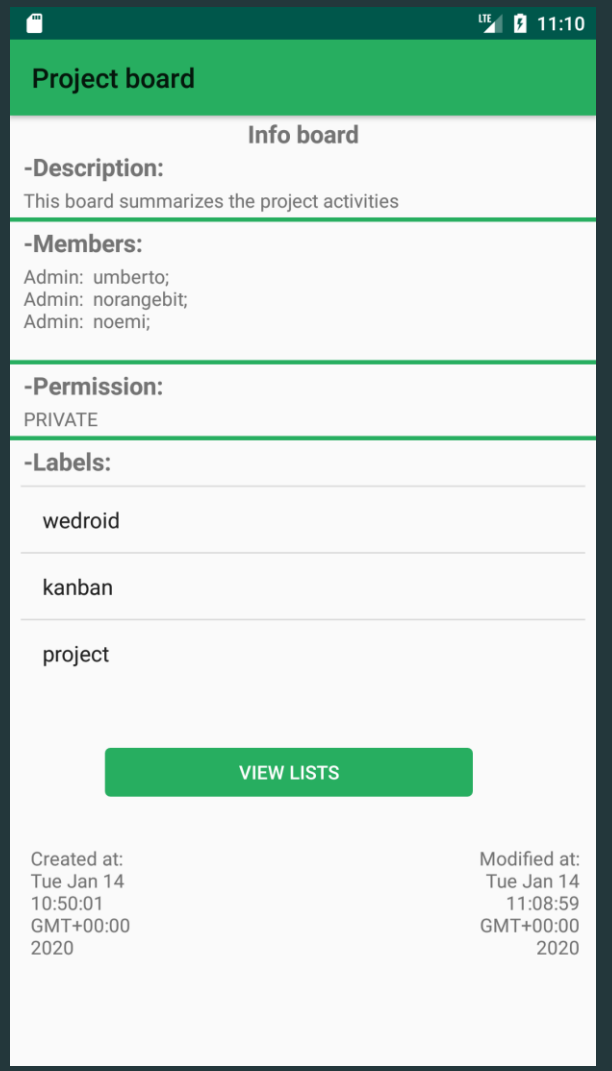
Funzionalità - Boards List

Visualizzazione delle boards dell’utente



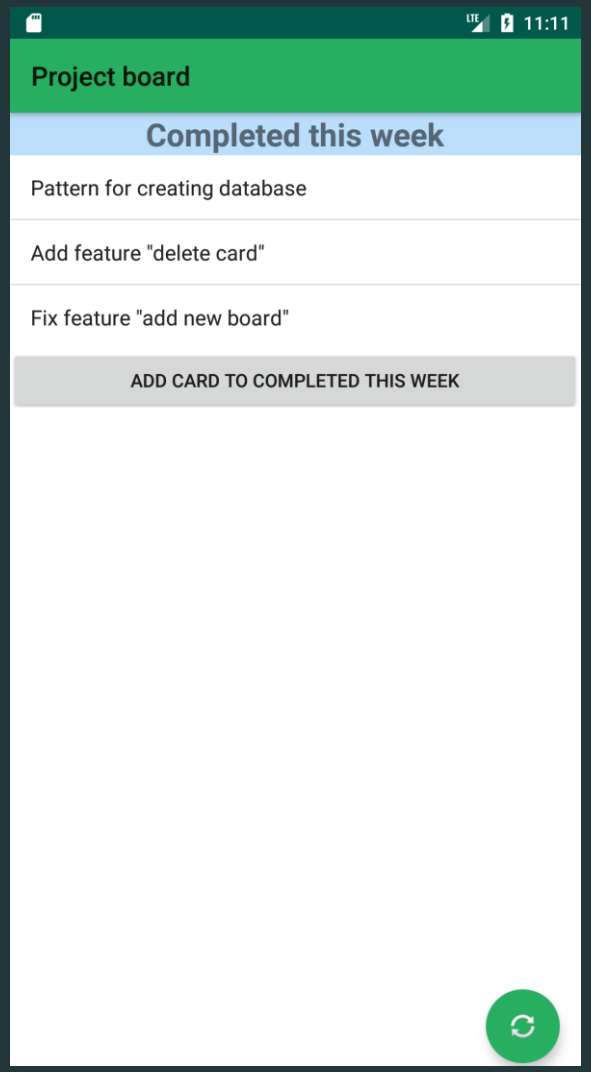
Funzionalità - Board View

Visualizzazione delle principali informazioni della board selezionata



Funzionalità - List View

Activity per la gestione delle liste e corrispettive card



Licenza

The MIT License (MIT)

Permessi

- Distribuzione
- Modifiche
- Uso Commerciale
- Uso Privato

Limitazioni

- Garanzia
- Responsabilità

Testing

Room

I metodi dei **DAO** sono stati testati attraverso un approccio ***blackbox***.

Poiché essi sono delle *suspending functions* si è reso necessario l'utilizzo di un costruttore di *coroutine* bloccante.

Infine, per eseguire le asserzioni, è stato necessario l'utilizzo di un observer, in quanto i metodi fanno ricorso alla classe LiveData.

Room

```
@Test
fun insert() {
  val board = Board("id", "title")

  runBlocking {
    dao.insert(board)
  }

  dao.getAllBoard().observeOnce {
    assertEquals(1, it.size)
    assertEquals(board, it[0])
  }
}
```

Wrapper

Le classi per la comunicazione con il server sono state testate attraverso un approccio ***blackbox***.

In particolare si è verificato se le risposte del server venivano de-serializzate correttamente nelle classi del dominio.

Wrapper

```
@Test
public void getCardTest(){
  MockResponse response = new MockResponse()
    .setResponseCode(HttpURLConnection.HTTP_OK)
    .setBody(cardJson);
  mockWebServer.enqueue(response);

  Card card = service.getCard(...)
    .execute().body();

  assertNotNull(card);
  assertEquals("check id", "id", card.getId());
  assertEquals("check title", "title", card.getTitle());
}
```

Repository

Il testing del repository è avvenuto tramite un approccio ***whitebox***.

```
fun deleteBoard(id: String) {
  service.deleteBoard(id).enqueue(
    object : Callback<Void> {
      override fun onResponse(
        call: Call<Void>,
        response: Response<Void>
      ) {
        CoroutineScope(Dispatchers.IO).launch {
          deleteBoardCallback(response, id)
        }
      }
    }
  )
}
```

Repository

```
@Test
fun deleteBoardCallbackSuccess() {
    val dao = mockk<BoardDao>()
    val response = mockk<Response<Void>>()

    every { response.isSuccessful } returns true
    coEvery { dao.delete(any()) } answers {}

    val deleteBoard = getPrivateFun(
        "deleteBoardCallback", BoardRepository::class
    )
```

Repository

```
    val repository = BoardRepository(
      dao, service, reader
    )

    runBlocking {
      deleteBoard?.callSuspend(
        repository,
        response,
        "id"
      )
    }

    coVerify { dao.delete(Board("id")) }
}
```

UI - Category Partition

Login

- instance
  - empty [error]
  - unformed [error]
  - formed [ok]
- username
  - empty [error]
  - not empty [ok]
- password
  - empty [error]
  - not empty [ok]

UI - Espresso

```
@Test
fun loginWithEmptyUrl() {
  onView(withId(R.id.username))
    .perform(typeText("username"))
  onView(withId(R.id.password))
    .perform(typeText("password"))
  closeSoftKeyboard()
  onView(withId(R.id.button))
    .perform(click())
  onView(withText("Riempire tutti i campi"))
    .inRoot(withDecorView(
      not(activityRule.activity.window.decorView)
    ))
    .check(matches(isDisplayed()))
}
```

Coverage

debugAndroidTest

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
it.unisannio.ding.ids.wedroid.app.data.dao		87%		62%	6	38	10	100	0	30	0	10
it.unisannio.ding.ids.wedroid.app.data.database		40%		29%	30	54	70	140	10	30	0	6
it.unisannio.ding.ids.wedroid.app.data.entity		58%		41%	21	32	0	8	9	20	0	4
it.unisannio.ding.ids.wedroid.app.data.repository		60%		51%	40	82	34	93	26	62	4	17
it.unisannio.ding.ids.wedroid.app.util		85%		100%	7	37	11	56	7	36	0	11
it.unisannio.ding.ids.wedroid.app.view		80%		50%	56	119	76	305	15	66	0	19
it.unisannio.ding.ids.wedroid.app.view.adapter		92%		53%	15	51	10	104	5	38	0	13
it.unisannio.ding.ids.wedroid.app.viewmodel		69%		0%	4	11	8	30	2	9	0	2
Total	1,481 of 5,436	72%	139 of 261	46%	179	424	219	836	74	291	4	82

Coverage

```
112.
113.         val title = data.getStringExtra(NewBoardActivity.BOARD_NAME)
114.         if (title == null) {
115.             Toast.makeText(this, R.string.on_null_new_board_name, Toast.LENGTH_LONG)
116.                 .show()
117.             return
118.         }
```

```
85.             override fun onResponse(
86.                 call: Call<it.unisannio.ding.ids.wedroid.wrapper.entity.Board>,
87.                 response: Response<it.unisannio.ding.ids.wedroid.wrapper.entity.Board>
88.             ) {
89.                 CoroutineScope(Dispatchers.IO).launch {
90.                     insertBoardCallback(response, title)
91.                 }
92.             }
93.         })
```

Some data

152 Commits
12 Issues
19 Pull Requests
153 Build